STA6706 Project 1

Robert Norberg

10/1/2014

Problem 1

We will consider the "Boston" housing data set, from the "MASS" library in R.

```
library(MASS) # load namespace `MASS`
data(Boston) # the object "Boston" now appears in the global environment
```

1. Based on this data set, provide an estimate for the population coefficient of variation, σ/μ, of "medv". Call this estimate $\widehat{\sigma}/\widehat{\mu}$.

First I compute the sample standard deviation and sample mean, then the coefficient of variation $\widehat{\sigma}/\widehat{\mu}$ from these estimates.

```
s <- sd(Boston$medv) # sample std dev
xbar <- mean(Boston$medv) # arithmetic mean
mycv <- s/xbar # coefficient of variation
mycv
```

```
[1] 0.4082
```

2. Propose an algorithm to obtain the standard normal bootstrap confidence interval. Use the algorithm proposed to compute a 95% standard normal bootstrap confidence interval for the population coefficient of variation, σ/μ, of "medv".

With several bootstrap replicate samples $b = 1, ..., B$, the normal bootstrap confidence interval is given by

$$\widehat{\theta} \pm z_{\alpha/2} se(\widehat{\theta}),$$

where $se(\widehat{\theta})$ is estimated by bootstrapping. This estimate is

$$\widehat{se}(\widehat{\theta}^*) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\widehat{\theta}^{(b)} - \overline{\widehat{\theta}}^*)^2}$$

and $\overline{\widehat{\theta}}^* = \frac{1}{B}\sum_{b=1}^{B}\widehat{\theta}^{(b)}$.

Given these formulations, we propose the following algorithm for computing the 95% standard normal bootstrap confidence interval:

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{*(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.
2. Compute $\widehat{\theta}^{(b)}$ for the b^{th} bootstrap sample.

3. Compute $\overline{\widehat{\theta}}^* = \frac{1}{B}\sum_{b=1}^{B}\widehat{\theta}^{(b)}$ from all of the computed $\widehat{\theta}^{(b)}$'s.

4. Use the $\widehat{\theta}^{(b)}$'s and $\overline{\widehat{\theta}}^*$ to compute $\widehat{se}(\widehat{\theta}^*) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\widehat{\theta}^{(b)} - \overline{\widehat{\theta}}^*)^2}$

5. With an estimate of the standard error for $\widehat{\theta}$ now in hand, compute the $100(1-\alpha)\%$ confidence interval for θ, $\widehat{\theta} \pm z_{\alpha/2}se(\widehat{\theta})$.

We demonstrate this algorithm by using it to compute the 95% standard normal bootstrap confidence interval for the population coefficient of variation, $\widehat{\sigma}/\widehat{\mu}$, of "medv".

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{*(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.

```
B <- 1000 # we will do 1000 bootstrap replicates
n <- length(Boston$medv) # number of obs to be in each replicate
set.seed(8675309) # set random seed for repeatability
# make B replicate samples and place them in a list object
my_replicates <- lapply(1:B, function(x) sample(Boston$medv, size=n, replace=T))
```

2. Compute $\widehat{\theta}^{(b)}$ for the b^{th} bootstrap sample.

```
theta_hats <- sapply(my_replicates, function(x) sd(x)/mean(x)) # coef of variation for each replicate
```

3. Compute $\overline{\widehat{\theta}}^* = \frac{1}{B}\sum_{b=1}^{B}\widehat{\theta}^{(b)}$ from all of the computed $\widehat{\theta}^{(b)}$'s.

```
theta_hat_bar <- mean(theta_hats)
theta_hat_bar
```

```
[1] 0.4065
```

4. Use the $\widehat{\theta}^{(b)}$'s and $\overline{\widehat{\theta}}^*$ to compute $\widehat{se}(\widehat{\theta}^*) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\widehat{\theta}^{(b)} - \overline{\widehat{\theta}}^*)^2}$.

```
theta_hat_errors <- theta_hats-theta_hat_bar
theta_hat_sq_errors <- theta_hat_errors^2
std_err_theta <- sqrt( (1/(B-1)) * sum(theta_hat_sq_errors) )
std_err_theta
```

```
[1] 0.01383
```

5. With an estimate of the standard error for $\widehat{\theta}$ now in hand, compute the $100(1-\alpha)\%$ confidence interval for θ, $\widehat{\theta} \pm z_{\alpha/2}se(\widehat{\theta})$.

```
alpha <- 0.05
z <- qnorm(alpha/2, lower.tail=F)
lower <- mycv-z*std_err_theta
upper <- mycv+z*std_err_theta
c(lower, upper)
```

```
[1] 0.3811 0.4353
```

So we can say with 95% confidence that the true population coefficient of variation is between 0.3811 and 0.4353.

3. Propose an algorithm to obtain a bootstrap t confidence interval. Use the algorithm proposed to compute a bootstrap t confidence interval for the population coefficient of variation, σ/μ, of "medv" with confidence level 95%.

The $100(1-\alpha)\%$ bootstrap t confidence interval is

$$(\widehat{\theta} - t^*_{1-\alpha/2}\widehat{se}(\widehat{\theta}),\ \widehat{\theta} + t^*_{1-\alpha/2}\widehat{se}(\widehat{\theta})$$

where $\widehat{se}(\widehat{\theta})$, $t^*_{\alpha/2}$, and $t^*_{1-\alpha/2}$ are defined as below.

We propose the following algorithm for computing the 95% t confidence interval.

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.
2. For each replicate b_i,

(a) Compute $\widehat{\theta}^{(b)}$ from the b^{th} sample $x^{(b)}$
(b) Estimate the standard error of $\widehat{\theta}^{(b)}$, $\widehat{se}(\widehat{\theta}^{(b)})$ by resampling the b^{th} bootstrap sample.
(c) Compute the b^{th} replicate of the "t" statistic, $t^{(b)} = \dfrac{\widehat{\theta}^{(b)} - \widehat{\theta}}{\widehat{se}(\widehat{\theta}^{(b)})}$.

3. From the sample of replicates $t^{(1)}, ..., t^{(B)}$ (which will be the reference distribution for bootstrap t), find the sample quantiles $t^*_{\alpha/2}$ and $t^*_{1-\alpha/2}$.
4. Compute $\widehat{se}(\widehat{\theta})$, the sample standard deviation of the replicates $\widehat{\theta}^{(b)}$.
5. Compute the confidence limits
$$(\widehat{\theta} - t^*_{1-\alpha/2}\widehat{se}(\widehat{\theta}),\ \widehat{\theta} - t^*_{\alpha/2}\widehat{se}(\widehat{\theta})$$

We demonstrate this algorithm by using it to compute the 95% t bootstrap confidence interval for the population coefficient of variation, $\widehat{\sigma}/\widehat{\mu}$, of "medv".

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.

We will reuse the bootstrap samples created in the previous problem, stored in the list `my_replicates`.

2. For each replicate b_i,

(a) Compute $\widehat{\theta}^{(b)}$ from the b^{th} sample $x^{(b)}$
(b) Estimate the standard error of $\widehat{\theta}^{(b)}$, $\widehat{se}(\widehat{\theta}^{(b)})$ by resampling the b^{th} bootstrap sample.
(c) Compute the b^{th} replicate of the "t" statistic, $t^{(b)} = \dfrac{\widehat{\theta}^{(b)} - \widehat{\theta}}{\widehat{se}(\widehat{\theta}^{(b)})}$.

Each $\widehat{\theta}^{(b)}$ is already calculated and stored in the vector `theta_hats`.

To find the bootstrap standard error for each bootstrap replicate, we will define a function `bootstrap_se()` and apply it to each bootstrap sample.

```r
# define function to take bootstrap standard error
bootstrap_se <- function(x, num_replicates, estimate_fun){
  # sample size
  n <- length(x)
  # pre-allocate empty vector for bootstrap estimates
  boot_ests <- vector(mode='numeric', length=num_replicates)
  # take bootstrap samples and find estimate of each
  for(i in 1:num_replicates){
    boot_ests[i] <- estimate_fun(sample(x, size=n, replace=T))
  }
  # find mean of bootstrap estimates
  boot_est_mean <- mean(boot_ests)
  # find squared error of each bootstrap estimate
  boot_sq_errs <- (boot_ests-boot_est_mean)^2
  # find std error
  boot_std_err <- sqrt( (1/(num_replicates-1)) * sum(boot_sq_errs) )

  return(boot_std_err)
}

# define coef_var function to pass to the estimate_fun argument of bootstrap_se()
coef_var <- function(x) sd(x)/mean(x)

# apply bootstrap_se() to each bootstrap sample in my_replicates
se_hats <- sapply(my_replicates, bootstrap_se, num_replicates=200, estimate_fun=coef_var)
```

Now that we have $\widehat{\theta}^{(b)}$ and $\widehat{se}(\widehat{\theta}^{(b)})$ for each replicate, we can calculate $t^{(b)}$ for each replicate (reusing the vector of bootstrap estimates `theta_hats` from earlier).

```r
my_ts <- (theta_hats-mycv)/(se_hats)
```

3. From the sample of replicates $t^{(1)}$, ..., $t^{(B)}$ (which will be the reference distribution for bootstrap t), find the sample quantiles $t^*_{\alpha/2}$ and $t^*_{1-\alpha/2}$.

```r
t_lower <- quantile(my_ts, (1-alpha/2))
t_upper <- quantile(my_ts, alpha/2)
c(t_lower, t_upper)
```

```
 97.5%   2.5%
 1.801 -2.202
```

4. Compute $\widehat{se}(\widehat{\theta})$, the sample standard deviation of the replicates $\widehat{\theta}^{(b)}$.

We will reuse the value calculated earlier, `std_err_theta`.

5. Compute the confidence limits

$$(\widehat{\theta} - t^*_{1-\alpha/2}\widehat{se}(\widehat{\theta}),\ \widehat{\theta} - t^*_{\alpha/2}\widehat{se}(\widehat{\theta}))$$

```
lower <- mycv-t_lower*std_err_theta
upper <- mycv-t_upper*std_err_theta
c(lower, upper)
```

```
 97.5%   2.5%
0.3833 0.4386
```

4. Propose an algorithm to obtain a BCa bootstrap confidence interval. Use the algorithm proposed to compute a 95% BCa bootstrap confidence interval for the population coefficient of variation, σ/μ, of "medv".

The BCa bootstrap confidence interval uses the α_1^{th} and the α_2^{th} percentiles of the distribution of $\widehat{\theta}^*$, where

$$\alpha_1 = \Phi\left(\widehat{z}_0 + \frac{\widehat{z}_0 + z_{\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{\alpha/2})}\right),$$

$$\alpha_2 = \Phi\left(\widehat{z}_0 + \frac{\widehat{z}_0 + z_{1-\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{1-\alpha/2})}\right)$$

z_0 is a correction for bias and is estimated by $\widehat{z}_0 = \Phi^{-1}\left(\frac{1}{B}\sum_{b=1}^{B} I(\widehat{\theta}^{(b)} < \widehat{\theta})\right)$.

$\widehat{a}$ is a correction for skewness and is estimated by $\widehat{a} = \frac{\sum_{i=1}^{n}(\overline{\theta_{(.)}} - \theta_{(i)})^3}{6\sum_{i=1}^{n}((\overline{\theta_{(.)}} - \theta_{(i)})^2)^{3/2}}$.

$\theta_(i)$ is the estimate computed from a sample missing the i^{th} point.

$\overline{\widehat{\theta}_{(.)}}$ is the mean of the estimates calculated from leave-one-out samples, given by $\overline{\widehat{\theta}_{(.)}} = \frac{1}{n}\sum_{i=1}^{n}\widehat{\theta}_{(i)}$.

Using these formulations, we propose the following algorithm for computing the BCa bootstrap confidence interval:

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.
2. For each replicate b_i, compute $\widehat{\theta}^{(b)}$ from the b^{th} sample $x^{(b)}$.
3. From the B calculated $\widehat{\theta}s$, calculate $\widehat{z}_0 = \Phi^{-1}\left(\frac{1}{B}\sum_{b=1}^{B} I(\widehat{\theta}^{(b)} < \widehat{\theta})\right)$.
4. For each leave-one-out sample $x_{(i)} = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$, compute $\widehat{\theta}_{(i)}$.
5. Compute $\overline{\widehat{\theta}_{(.)}}$, the mean of all n computed $\widehat{\theta}_{(i)}s$.
6. From the n computed $\widehat{\theta}_{(i)}s$ and $\overline{\widehat{\theta}_{(.)}}$, compute $\widehat{a} = \frac{\sum_{i=1}^{n}(\overline{\theta_{(.)}} - \theta_{(i)})^3}{6\sum_{i=1}^{n}((\overline{\theta_{(.)}} - \theta_{(i)})^2)^{3/2}}$.
7. Using $\widehat{a}$ and $\widehat{z}_0$, compute $\alpha_1 = \Phi\left(\widehat{z}_0 + \frac{\widehat{z}_0 + z_{\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{\alpha/2})}\right)$ and $\alpha_2 = \Phi\left(\widehat{z}_0 + \frac{\widehat{z}_0 + z_{1-\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{1-\alpha/2})}\right)$.
8. Find the α_1^{th} and α_2^{th} percentiles of the distribution of $\widehat{\theta}^*$.

We demonstrate this algorithm by using it to compute the 95% BCa bootstrap confidence interval for the population coefficient of variation, $\widehat{\sigma}/\widehat{\mu}$, of "medv".

1. For each bootstrap replicate $b = 1, ..., B$, generate n random integers $\{i_1, i_2, ..., i_n\}$ uniformly on the set $\{1, 2, ..., n\}$ and select the bootstrap sample $x^{(b)} = (x_{i1}, x_{i2}, ..., x_{in})$.

We will reuse the bootstrap samples taken earlier, stored in the list `my_replicates`.

2. For each replicate b_i, compute $\widehat{\theta}^{(b)}$ from the b^{th} sample $x^{(b)}$.

These are also already computed and stored in the vector `theta_hats`.

3. From the B calculated $\widehat{\theta}s$, calculate $\widehat{z}_0 = \Phi^{-1}\left(\frac{1}{B}\sum_{b=1}^{B} I(\widehat{\theta}^{(b)} < \widehat{\theta})\right)$.

This is just a count of how many $\widehat{\theta}^{(b)}s$ are greater that $\widehat{\theta}$ ($\widehat{\theta}$ is stored as `mycv`) divided by the total number of replicates, then passed to the inverse normal CDF.

```
z_0hat <- qnorm(sum(theta_hats>mycv)/B)
z_0hat
```

```
[1] -0.07276
```

4. For each leave-one-out sample $x_{(i)} = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$, compute $\widehat{\theta}_{(i)}$.

First we pre-allocate an empty vector for each $\widehat{\theta}_{(i)}$. There will be n of these. Then we remove the i^{th} value from the sample, one at a time, and calculate the coefficient of variation $\widehat{\theta}_{(i)}$, placing it into the i^{th} element of the pre-allocated vector. Note that `n` is stored from earlier and it is equal to the number of observations in the sample.

```
loo_theta_hats <- vector(mode='numeric', length=n)
for(i in 1:n){
  loo_theta_hats[i] <- coef_var(Boston$medv[-i])
}
```

5. Compute $\overline{\widehat{\theta}_{(.)}}$, the mean of all n computed $\widehat{\theta}_{(i)}s$.

We just take the mean of the computed leave-one-out $\widehat{\theta}_{(i)}s$.

```
loo_theta_hats_mean <- mean(loo_theta_hats)
```

6. From the n computed $\widehat{\theta}_{(i)}s$ and $\overline{\widehat{\theta}_{(.)}}$, compute $\widehat{a} = \frac{\sum_{i=1}^{n}(\overline{\theta_{(.)}} - \theta_{(i)})^3}{6\sum_{i=1}^{n}((\overline{\theta_{(.)}} - \theta_{(i)})^2)^{3/2}}$.

First we calculate just the numerator of $\widehat{a}$, $\sum_{i=1}^{n}(\overline{\theta_{(.)}} - \theta_{(i)})^3$.

```
loo_thete_hat_errs <- loo_theta_hats_mean-loo_theta_hats
numerator <- sum((loo_thete_hat_errs)^3)
```

Then the denominator, $6\sum_{i=1}^{n}((\overline{\theta_{(.)}} - \theta_{(i)})^2)^{3/2}$.

```
denominator <- 6*sum(loo_thete_hat_errs^2)^(3/2)
```

Then finally calculate $\widehat{a}$.

```
ahat <- numerator/denominator
ahat
```

```
[1] 0.01547
```

7. Using $\widehat{a}$ and $\widehat{z}_0$, compute $\alpha_1 = \Phi\left(\widehat{z}_0 + \dfrac{\widehat{z}_0 + z_{\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{\alpha/2})}\right)$ and $\alpha_2 = \Phi\left(\widehat{z}_0 + \dfrac{\widehat{z}_0 + z_{1-\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{1-\alpha/2})}\right)$.

First we compute $\widehat{z}_0 + z_{\alpha/2}$, then use this value to compute $\dfrac{\widehat{z}_0 + z_{\alpha/2}}{1 - \widehat{a}(\widehat{z}_0 + z_{\alpha/2})}$, the finally pass this value to the standard normal CDF to find α_1.

```
tmp <- z_0hat+qnorm(alpha/2)
tmp2 <- z_0hat+((tmp)/(1-(ahat*tmp)))
alpha_1 <- pnorm(tmp2)
alpha_1
```

```
[1] 0.0205
```

And we repeat this to find α_2, using $z_{1-\alpha_2}$ instead of $z_{\alpha/2}$.

```
tmp <- z_0hat+qnorm((1-(alpha/2)))
tmp2 <- z_0hat+((tmp)/(1-(ahat*tmp)))
alpha_2 <- pnorm(tmp2)
alpha_2
```

```
[1] 0.9693
```

8. Find the α_1^{th} and α_2^{th} percentiles of the distribution of $\widehat{\theta}^*$.

We use the `quantile()` function to find the α_1^{th} and α_2^{th} percentile of the B computed $\widehat{\theta}^* s$ (stored in the vector `theta_hats`).

```
quantile(theta_hats, c(alpha_1, alpha_2))
```

```
 2.05% 96.93%
0.3769 0.4318
```

Although these are not the 2.5^{th} and 97.5^{th} percentiles of the $\widehat{\theta}^* s$, this is the 95% BCa confidence interval for the true population coefficient of variation of "medv". This confidence interval is adjusted for the bias and skewness of the distribution of the $\widehat{\theta}^* s$.

Problem 2

The goal of problem 2 is to understand cross validation.

1. Read pages 175-183 of the textbook of James, Witten, Hastie, and Tibshirani and give a short summary.

Pages 175-183 of the textbook discuss validation. When fitting a model, one may asses the error rate of the model by looking at the model residuals or errors, but this can misrepresent the accuracy of the model when applied to new data. For this reason it is useful to hold out some data when fitting the model and then calculate the model's error rate when applied to these held out observations. The set of observations used to fit the model is called the "training" set and the held out observations are are called the "validation" set. The MSE of the validation set is most commonly used to asses the model's accuracy.

One may choose to divide the data into a training and a validation set, fit a model once (on the training set), and test the model on the validation set. Alternatively, one may choose to divide the data into several segments, fit a model on all but one, find the MSE of this validation set, then fit another model leaving out a different segment, find the MSE of this validation set, and so on, until each segment has been left out once. Then the mean of the MSE's is calculated, providing a better estimate of the validation set MSE than if only one validation set had been used. The data may be divided into any number of sets (call this number k). Then exactly k models must be fit and k validation set MSE's computed. This procedure is called k-fold cross-validation.

One may consider a special case of k-fold cross-validation where $k = n$. Then a model is fit n times (once for each observation) and used to predict the on left out observation. Then average squared error of each prediction. This procedure is called leave-one-out cross-validation.

2. Propose an algorithm for the k-fold cross validation.

1. Choose k, the number of segments you wish to divide the data into (and the number of models you will fit). The size of each data segment, call them S_1, S_2, ..., S_k, is $Size(S_*) = \frac{n}{k}$ (rounding down if necessary). There will be $n \mod k$ "leftover" observations.
2. Take the sequence of integers $1, ..., k$ and replicate it $Size(S_*)$ times. Append to this sequence the integers $1, ..., n \mod k$. This sequence will now be n integers in length. These integers are the segment assignments for for each of the n observations. To randomize the segment assignments, randomly permute the sequence of integers.
3. Fit a model using the data in sets S_2, ..., S_k (leaving out set S_1).
4. Use the model fit to predict the outcome of set S_1 and calculate the MSE of these predictions, MSE_1.
5. Repeat steps 3 and 4, leaving out S_2, then S_3, and so on until MSE_1, MSE_2, ..., MSE_k have been attained.
6. Find the mean of MSE_1, MSE_2, ..., MSE_k

3. In example 7.18 on page 210 of the textbook of Rizzo, LOOCV was used to select the best fitting model. Repeat the same analysis using a 10-fold cross validation. Compare the results using the two methods.

Problem 7.18 uses the "ironslag" data set contained in the `DAAG` package. We load the namespace of this package and call the data set into the global environment.

```
library(DAAG) # load library for ironslag data set
data(ironslag) # data now exists in the global work space
```

We demonstrate the algorithm for k-fold cross-validation laid out earlier to replicate example 7.18.

1. Choose k, the number of segments you wish to divide the data into (and the number of models you will fit). The size of each data segment, call them S_1, S_2, ..., S_k, is $Size(S_*) = \frac{n}{k}$ (rounding down if necessary). There will be $n \mod k$ "leftover" observations.

We wish to divide the data into 10 segments. The number of observations in the data is not evenly divisible by 10, so there will be some "leftover" observations. Note that the modulus operator in R is `%%`.

```
k <- 10 # for 10-fold cross-validation
n <- nrow(ironslag) # sample size
seg_size <- floor(n/k) # approx size of each segment
leftovers <- n%%k
```

2. Take the sequence of integers $1, ..., k$ each replicated $Size(S_*)$ times. If $n \mod k > 0$, append to this sequence the integers $1, ..., n \mod k$. This sequence will now be n integers in length. These integers are the segment assignments for for each of the n observations. To randomize the segment assignments, randomly permute the sequence of integers.

We generate the suggested sequence, and since we have some "leftover" points, we append those to the sequence. Once the sequence of integers is created, we permute it using the `sample()` function, being sure to set a random seed beforehand.

```
seg_assignments <- rep(1:k, each=seg_size) # create segment assignments
seg_assignments <- c(seg_assignments, 1:leftovers) # add "leftover" observations
set.seed(8675309) # set random seed
seg_assignments <- sample(seg_assignments) # randomly permute segment assignments
```

3. Fit a model using the data in sets S_2, ..., S_k (leaving out set S_1).

We fit the four models fit in example 7.18 for all observations not in Segment 1.

```
train <- ironslag[seg_assignments!=1, ]
lin_mod <- lm(magnetic~chemical, data=train) # Fit linear model using training data
quad_mod <- lm(magnetic~chemical+I(chemical^2), data=train ) # Fit quadratic model using training data
exp_mod <- lm(log(magnetic)~chemical, data=train) # Fit exponential model using training data
log_mod <- lm(log(magnetic)~log(chemical), data=train) # Fit log-log model using training data
```

4. Use the model fit to predict the outcome of set S_1 and calculate the MSE of these predictions, MSE_1.

We use the four models fit to predict the validation set, being sure to exponentiate the predictions made by the exponential and log-log models because those predictions will be in log-units.

```
validation <- ironslag[seg_assignments==1, ]
lin_pred <- predict(lin_mod, newdata=validation) # use linear model to predict validation set
quad_pred <- predict(quad_mod, newdata=validation) # use quadratic model to predict validation set
exp_pred <- exp(predict(exp_mod, newdata=validation)) # remember to exponentiate predictions
log_pred <- exp(predict(log_mod, newdata=validation)) # remember to exponentiate predictions
```

Then we find the errors of each prediction set and calculate MSE_1 for each model.

```
lin_errors <- (validation$chemical-lin_pred)
quad_errors <- (validation$chemical-quad_pred)
exp_errors <- (validation$chemical-exp_pred)
log_errors <- (validation$chemical-log_pred)

lin_MSE <- mean(lin_errors^2)
quad_MSE <- mean(quad_errors^2)
exp_MSE <- mean(exp_errors^2)
log_MSE <- mean(log_errors^2)

c(lin_MSE, quad_MSE, exp_MSE, log_MSE)
```

```
[1] 0.981 3.071 2.480 4.123
```

5. Repeat steps 3 and 4, leaving out S_2, then S_3, and so on until MSE_1, MSE_2, ..., MSE_k have been attained.

We first pre-allocate a data frame to store the four MSEs computed for each of the k training sets.

```
mse_df <- data.frame(lin_MSE=NA, quad_MSE=NA, exp_MSE=NA, log_MSE=NA)
```

Then we repeat steps 3 and 4 for each value of k, storing the computed MSEs in `mse_df` along the way.

```
for(i in 1:k){
  train <- ironslag[seg_assignments!=i, ] # define training set i

  # fit models for training set i
  lin_mod <- lm(magnetic~chemical, data=train) # linear model
  quad_mod <- lm(magnetic~chemical+I(chemical^2), data=train) # quadratic model
  exp_mod <- lm(log(magnetic)~chemical, data=train) # exponential model
  log_mod <- lm(log(magnetic)~log(chemical), data=train) # log-log model

  validation <- ironslag[seg_assignments==i, ] # define validation set i

  # make predictions for validation set i
  lin_pred <- predict(lin_mod, newdata=validation) # linear model predictions
  quad_pred <- predict(quad_mod, newdata=validation) # quadratic model predictions
  exp_pred <- exp(predict(exp_mod, newdata=validation)) # exponential model predictions
  log_pred <- exp(predict(log_mod, newdata=validation)) # log-log model predictions

  # calculate prediction errors for validation set i
  lin_errors <- (validation$chemical-lin_pred)
  quad_errors <- (validation$chemical-quad_pred)
  exp_errors <- (validation$chemical-exp_pred)
  log_errors <- (validation$chemical-log_pred)

  # calculate MSE for validation set i
  lin_MSE <- mean(lin_errors^2)
  quad_MSE <- mean(quad_errors^2)
  exp_MSE <- mean(exp_errors^2)
```

```
  log_MSE <- mean(log_errors^2)

  # store calculated MSEs in mse_df
  mse_df[i, ] <- c(lin_MSE, quad_MSE, exp_MSE, log_MSE)
  }
```

6. Find the mean of MSE_1, MSE_2, ..., MSE_k

We calculate the means of the four MSEs computed (which are stored in `mse_df`).

```
colMeans(mse_df)
```

```
 lin_MSE quad_MSE  exp_MSE  log_MSE
   0.531    2.877    1.598    2.256
```

Problem 3

Solve problem 7.8 on page 213 of the textbook of Rizzo.

The problem refers to the data set `scor` contained in the `bootstrap` package.

```
# load library `bootstrap` for the scor data set
library(bootstrap)
data(scor) # call the data set into the global environment
```

```
7.8 Refer to Excercise 7.7. Obtain the jacknife estimates of bias and standard error of $\widehat{\theta
```

Excercise 7.7 states:

The five-dimensional scores data have a 5 x 5 covariabnce matrix Σ, with positive eigenvalues $\lambda_1 > ... > \lambda_5$. In principal component analysis,

$$\theta = \frac{\lambda_1}{\sum_{j=1}^{5} \lambda_j}$$

measures the proportion of the variance explained by the first principal component. Let $\widehat{\lambda_1} > ... > \widehat{\lambda_5}$ be the eigenvalues of $\widehat{\Sigma}$, where $\widehat{\Sigma}$ is the MLE of Σ. Compute the sample estimate

$$\widehat{\theta} = \frac{\widehat{\lambda_1}}{\sum_{j=1}^{5} \widehat{\lambda_j}}$$

So we wish to obtain the jacknife estimates of bias and standard error of $\widehat{\theta}$ as given above, in reference to the `scor` data set.

Defining the i^{th} jacknife sample $x_{(i)}$ as the subset of the original data that leaves out the i^{th} observation x_i, the jacknife estimate of bias is

$$\widehat{bias}_{jack} = (n-1)(\overline{\widehat{\theta}}_{(.)} - \widehat{\theta}),$$

where $\bar{\widehat{\theta}}_{(.)} = \frac{1}{n}\sum_{i=1}^{n}\widehat{\theta}_{(i)}$ is the mean of the estimates of the leave-one-out samples, and $\widehat{\theta} = \widehat{\theta}(x)$ is the estimate computed from the original observed sample.

Since we will be computing $n+1$ $\widehat{\theta}$'s, it will simplify things to define a function that computes for any sample

$$\widehat{\theta} = \frac{\widehat{\lambda_1}}{\sum_{j=1}^{5}\widehat{\lambda}_j}$$

```
myfunc <- function(mysamp){
  sigma_hat <- cov(mysamp) # MLE of Sigma
  evals <- eigen(sigma_hat)$values # eigenvalues of Sigma hat
  lambda_1 <- max(evals) # largest eigenvalue
  lambda_sum <- sum(evals) # sum of all eigenvalues
  theta_hat <- lambda_1/lambda_sum # proportion of variance explained by first principal component
  return(theta_hat)
}
```

First, we compute $\widehat{\theta}$ from all observations x_1, x_2, ..., x_n using the function `myfunc()`.

```
my_theta_hat <- myfunc(scor) # find theta hat using all x's
my_theta_hat
```

```
[1] 0.6191
```

Next we calculate $\widehat{\theta}_{(i)}$ for $x_{(1)}$, $x_{(2)}$, ..., $x_{(n)}$ again using the function `myfunc()`.

```
theta_hat_vec <- vector(mode='numeric', length=nrow(scor)) # pre-allocate empty vector for theta hats
for(i in 1:nrow(scor)){
  samp_x_i <- scor[-i, ] # leave out observation i
  theta_hat_i <- myfunc(samp_x_i) # calculate theta hat without obs i, using `myfunc`
  theta_hat_vec[i] <- theta_hat_i # assign computed theta hat to ith element in "theta_hat_vec"
}
```

From these $\widehat{\theta}_{(i)}$'s we can now compute $\bar{\widehat{\theta}}_{(.)}$.

```
theta_hat_bar <- mean(theta_hat_vec) # mean of all theta hat i's
theta_hat_bar
```

```
[1] 0.6191
```

And finally we can compute the jacknife estimate of the bias of $\widehat{theta}$.

```
n <- nrow(scor) # sample size
jacknife_bias <- (n-1)*(theta_hat_bar-my_theta_hat) # jacknife est of bias
jacknife_bias
```

```
[1] 0.001069
```

To compute the jacknife estimate of standard error of $\widehat{\theta}$,

$$\widehat{se}_{jack} = \sqrt{\frac{n-1}{n}\sum(\widehat{\theta}_{(i)} - \overline{\widehat{\theta}}_{(.)})^2}$$

we can recycle the $\widehat{\theta}_{(i)}$'s computed earlier. First we calculate the error of each $\widehat{\theta}_{(i)}$ (the difference between it and $\overline{\widehat{\theta}}_{(.)}$).

```
theta_hat_errors <- theta_hat_vec-theta_hat_bar # error of each theta hat i
```

Then find the squared errors:

```
theta_hat_sq_errors <- theta_hat_errors^2 # squared errors from raw errors
```

And finally compute $\widehat{se}_{jack}$.

```
jacknife_se <- sqrt( ((n-1)/n) * sum(theta_hat_sq_errors) ) # compute jacknife se
jacknife_se
```

```
[1] 0.04955
```

Appendix with R code

```
# Clear working environment
rm(list=ls())

# Options for document compilation
knitr::opts_chunk$set(warning=FALSE, message=FALSE, comment=NA, fig.width=4, fig.height=3)
library(MASS) # load namespace `MASS`
data(Boston) # the object "Boston" now appears in the global environment
s <- sd(Boston$medv) # sample std dev
xbar <- mean(Boston$medv) # arithmetic mean
mycv <- s/xbar # coefficient of variation
mycv
B <- 1000 # we will do 1000 bootstrap replicates
n <- length(Boston$medv) # number of obs to be in each replicate
set.seed(8675309) # set random seed for repeatability
# make B replicate samples and place them in a list object
my_replicates <- lapply(1:B, function(x) sample(Boston$medv, size=n, replace=T))
theta_hats <- sapply(my_replicates, function(x) sd(x)/mean(x)) # coef of variation for each replicate
theta_hat_bar <- mean(theta_hats)
theta_hat_bar
theta_hat_errors <- theta_hats-theta_hat_bar
theta_hat_sq_errors <- theta_hat_errors^2
std_err_theta <- sqrt( (1/(B-1)) * sum(theta_hat_sq_errors) )
std_err_theta
alpha <- 0.05
z <- qnorm(alpha/2, lower.tail=F)
lower <- mycv-z*std_err_theta
```

```
upper <- mycv+z*std_err_theta
c(lower, upper)
# define function to take bootstrap standard error
bootstrap_se <- function(x, num_replicates, estimate_fun){
  # sample size
  n <- length(x)
  # pre-allocate empty vector for bootstrap estimates
  boot_ests <- vector(mode='numeric', length=num_replicates)
  # take bootstrap samples and find estimate of each
  for(i in 1:num_replicates){
    boot_ests[i] <- estimate_fun(sample(x, size=n, replace=T))
  }
  # find mean of bootstrap estimates
  boot_est_mean <- mean(boot_ests)
  # find squared error of each bootstrap estimate
  boot_sq_errs <- (boot_ests-boot_est_mean)^2
  # find std error
  boot_std_err <- sqrt( (1/(num_replicates-1)) * sum(boot_sq_errs) )

  return(boot_std_err)
}

# define coef_var function to pass to the estimate_fun argument of bootstrap_se()
coef_var <- function(x) sd(x)/mean(x)

# apply bootstrap_se() to each bootstrap sample in my_replicates
se_hats <- sapply(my_replicates, bootstrap_se, num_replicates=200, estimate_fun=coef_var)
my_ts <- (theta_hats-mycv)/(se_hats)
t_lower <- quantile(my_ts, (1-alpha/2))
t_upper <- quantile(my_ts, alpha/2)
c(t_lower, t_upper)
lower <- mycv-t_lower*std_err_theta
upper <- mycv-t_upper*std_err_theta
c(lower, upper)
z_0hat <- qnorm(sum(theta_hats>mycv)/B)
z_0hat
loo_theta_hats <- vector(mode='numeric', length=n)
for(i in 1:n){
  loo_theta_hats[i] <- coef_var(Boston$medv[-i])
}
loo_theta_hats_mean <- mean(loo_theta_hats)
loo_thete_hat_errs <- loo_theta_hats_mean-loo_theta_hats
numerator <- sum((loo_thete_hat_errs)^3)
denominator <- 6*sum(loo_thete_hat_errs^2)^(3/2)
ahat <- numerator/denominator
ahat
tmp <- z_0hat+qnorm(alpha/2)
tmp2 <- z_0hat+((tmp)/(1-(ahat*tmp)))
alpha_1 <- pnorm(tmp2)
alpha_1
tmp <- z_0hat+qnorm((1-(alpha/2)))
tmp2 <- z_0hat+((tmp)/(1-(ahat*tmp)))
alpha_2 <- pnorm(tmp2)
```

```
alpha_2
quantile(theta_hats, c(alpha_1, alpha_2))
library(DAAG) # load library for ironslag data set
data(ironslag) # data now exists in the global work space
k <- 10 # for 10-fold cross-validation
n <- nrow(ironslag) # sample size
seg_size <- floor(n/k) # approx size of each segment
leftovers <- n%%k
seg_assignments <- rep(1:k, each=seg_size) # create segment assignments
seg_assignments <- c(seg_assignments, 1:leftovers) # add "leftover" observations
set.seed(8675309) # set random seed
seg_assignments <- sample(seg_assignments) # randomly permute segment assignments
train <- ironslag[seg_assignments!=1, ]
lin_mod <- lm(magnetic~chemical, data=train) # Fit linear model using training data
quad_mod <- lm(magnetic~chemical+I(chemical^2), data=train ) # Fit quadratic model using training data
exp_mod <- lm(log(magnetic)~chemical, data=train) # Fit exponential model using training data
log_mod <- lm(log(magnetic)~log(chemical), data=train) # Fit log-log model using training data
validation <- ironslag[seg_assignments==1, ]
lin_pred <- predict(lin_mod, newdata=validation) # use linear model to predict validation set
quad_pred <- predict(quad_mod, newdata=validation) # use quadratic model to predict validation set
exp_pred <- exp(predict(exp_mod, newdata=validation)) # remember to exponentiate predictions
log_pred <- exp(predict(log_mod, newdata=validation)) # remember to exponentiate predictions
lin_errors <- (validation$chemical-lin_pred)
quad_errors <- (validation$chemical-quad_pred)
exp_errors <- (validation$chemical-exp_pred)
log_errors <- (validation$chemical-log_pred)

lin_MSE <- mean(lin_errors^2)
quad_MSE <- mean(quad_errors^2)
exp_MSE <- mean(exp_errors^2)
log_MSE <- mean(log_errors^2)

c(lin_MSE, quad_MSE, exp_MSE, log_MSE)
mse_df <- data.frame(lin_MSE=NA, quad_MSE=NA, exp_MSE=NA, log_MSE=NA)
for(i in 1:k){
  train <- ironslag[seg_assignments!=i, ] # define training set i

  # fit models for training set i
  lin_mod <- lm(magnetic~chemical, data=train) # linear model
  quad_mod <- lm(magnetic~chemical+I(chemical^2), data=train) # quadratic model
  exp_mod <- lm(log(magnetic)~chemical, data=train) # exponential model
  log_mod <- lm(log(magnetic)~log(chemical), data=train) # log-log model

  validation <- ironslag[seg_assignments==i, ] # define validation set i

  # make predictions for validation set i
  lin_pred <- predict(lin_mod, newdata=validation) # linear model predictions
  quad_pred <- predict(quad_mod, newdata=validation) # quadratic model predictions
  exp_pred <- exp(predict(exp_mod, newdata=validation)) # exponential model predictions
  log_pred <- exp(predict(log_mod, newdata=validation)) # log-log model predictions

  # calculate prediction errors for validation set i
  lin_errors <- (validation$chemical-lin_pred)
```

```
  quad_errors <- (validation$chemical-quad_pred)
  exp_errors <- (validation$chemical-exp_pred)
  log_errors <- (validation$chemical-log_pred)

  # calculate MSE for validation set i
  lin_MSE <- mean(lin_errors^2)
  quad_MSE <- mean(quad_errors^2)
  exp_MSE <- mean(exp_errors^2)
  log_MSE <- mean(log_errors^2)

  # store calculated MSEs in mse_df
  mse_df[i, ] <- c(lin_MSE, quad_MSE, exp_MSE, log_MSE)
  }
colMeans(mse_df)
# load library `bootstrap` for the scor data set
library(bootstrap)
data(scor) # call the data set into the global environment
myfunc <- function(mysamp){
  sigma_hat <- cov(mysamp) # MLE of Sigma
  evals <- eigen(sigma_hat)$values # eigenvalues of Sigma hat
  lambda_1 <- max(evals) # largest eigenvalue
  lambda_sum <- sum(evals) # sum of all eigenvalues
  theta_hat <- lambda_1/lambda_sum # proportion of variance explained by first principal component
  return(theta_hat)
}
my_theta_hat <- myfunc(scor) # find theta hat using all x's
my_theta_hat
theta_hat_vec <- vector(mode='numeric', length=nrow(scor)) # pre-allocate empty vector for theta hats
for(i in 1:nrow(scor)){
  samp_x_i <- scor[-i, ] # leave out observation i
  theta_hat_i <- myfunc(samp_x_i) # calculate theta hat without obs i, using `myfunc`
  theta_hat_vec[i] <- theta_hat_i # assign computed theta hat to ith element in "theta_hat_vec"
}
theta_hat_bar <- mean(theta_hat_vec) # mean of all theta hat i's
theta_hat_bar
n <- nrow(scor) # sample size
jacknife_bias <- (n-1)*(theta_hat_bar-my_theta_hat) # jacknife est of bias
jacknife_bias
theta_hat_errors <- theta_hat_vec-theta_hat_bar # error of each theta hat i
theta_hat_sq_errors <- theta_hat_errors^2 # squared errors from raw errors
jacknife_se <- sqrt( ((n-1)/n) * sum(theta_hat_sq_errors) ) # compute jacknife se
jacknife_se
```
